Coordination of AI Workloads and Telecom Infrastructures: The Way to Sustainability

Luis Velasco*, Pol González, Jaume Comellas and Marc Ruiz
Optical Communications Group (GCO), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
**e-mail: luis.velasco@upc.edu*

ABSTRACT
Federated Learning (FL) is a decentralized, collaborative Machine Learning technique to train Artificial Intelligence (AI) models directly on edge devices or systems in customer premises. Because FL preserves privacy and security, many applications of FL that require keeping sensitive information locally can be found in the literature, e.g., for clinical trials and autonomous driving. The locally trained models are periodically transferred to a centralized server, where global models are computed by combining the local ones, and the resulting model is distributed back to the devices/systems. It is precisely because of the high-capacity connectivity needed to transfer large models, e.g., Large Language Models (LLM) involving trillions of parameters, that the coordination of FL applications and telecom infrastructures is a must. In this paper, we propose workflows to manage the connectivity capacity by extending coordination between FL applications and the telecom cloud orchestrator.

Keywords: Federated Learning, Networking for AI

1. Introduction

Federated Learning (FL) is a privacy-preserving Machine Learning (ML) technique where multiple FL clients, coordinated with one or multiple central FL servers, collaborate to train a ML model [1]. Different from conventional centralized ML, where data is collected from different sources and conveyed to a central server for processing, data is used locally to train ML models that are exchanged between the FL clients and the centralized FL servers. Because of their characteristics, many applications of FL have been reported, including healthcare and users' behavior (see, e.g., [2], [3]). The possibility of using hardware accelerators available in the cloud owned by hyperscalers will make possible to train Large Language Models (LLM) for a fraction of the cost of developing such infrastructure internally. However, the cost of transferring the models between locations can be too high, as those LLMs might consist of trillions of parameters and occupy hundreds of GB to TB. This fact makes that telecom operators, owning both the telecom network and a cloud infrastructure, can offer competitive and efficient solutions.

In FL, local models are trained by the FL clients using local data. Once trained, the models are transferred to the FL server, which in turn, trains a global model and sends it back to the FL clients. This loop is repeated for several rounds, where the accuracy of the global model improves at every round. For FL to converge in practical times, not only the training time has to be accelerated, but also time to transfer the models. Therefore, in the case of training large LLMs, high-capacity connectivity between FL clients and the FL server is needed. In view of that, some authors have proposed solutions for making FL communications more efficient (see, e.g., [4]). Furthermore, connectivity capacity can be greatly reduced once the models are transferred and activated back after new models are produced. Such process clearly motivates the collaboration between FL applications and the telecom infrastructure beyond the availability of automatic interfaces for the deployment of FL workloads.

In this paper, we propose the collaboration between FL applications and the telecom operators to reduce, as much as possible, the usage of networking resources. Specifically, we propose coordinated orchestration and operation for on-demand network capacity allocation. In addition, we target at minimizing the number of models that are transferred at every round without major impact on the performance of the models. The rest of the paper is organized as follows. Section 2 presents the targeted scenario and a simplified control and orchestration architecture is proposed and defines the needed coordination at the orchestration level to optimize networking resources during the FL operation. Section 3 reports the experimental results of the implemented workflows. Finally, section 4 draws the main conclusions.

2. Federated Learning Orchestration and Operation

Figure 1 presents the scenario for a FL application. We assume that FL clients run in a customer private cloud infrastructure (locations 1 to *n* in Figure 1), e.g., hospitals in a healthcare FL application, whereas the FL server runs in a metro / metro-core location belonging to the telecom operator cloud. Client locations are connected to the telecom cloud infrastructure through an access/metro network. In the control/orchestration plane, a Customer Orchestrator is in charge of the deployment and operation of FL workloads on the targeted hybrid cloud infrastructure. To that end, it coordinates the deployment of workloads on customer cloud sites, e.g., configured as Kubernetes clusters, as well

979-8-3195-4420-9/26/$31.00

as on telecom cloud sites through the Telecom Orchestrator's north-bound interface. The latter coordinates the deployment of the customer workloads and manages the connectivity between the different locations through the access/metro network by means of a Software-Defined Networking (SDN) Controller.

In the case of FL applications training large models, exchanging local models from all the FL clients at every iteration requires high-capacity connectivity so that local models are transferred in practical times to the central FL server. However, once models are transferred, such capacity remains unused until the next FL round. Therefore, coordination between the FL application and the orchestration system would be beneficial to implement capacity-on-demand approaches ensuring that connectivity capacity is available when it is really needed by the application, while making unneeded capacity resources available for other services.

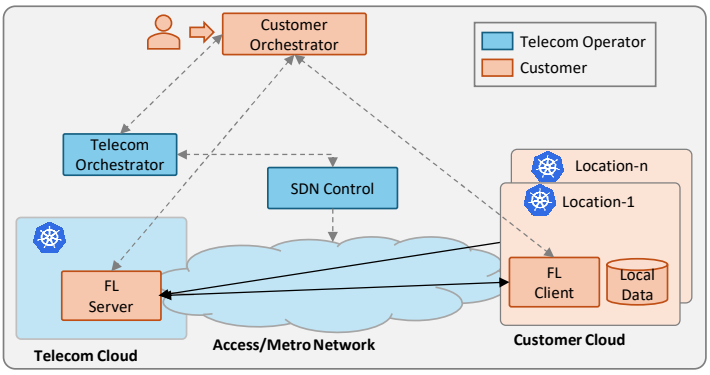

Figure 1 Federated Learning Scenario

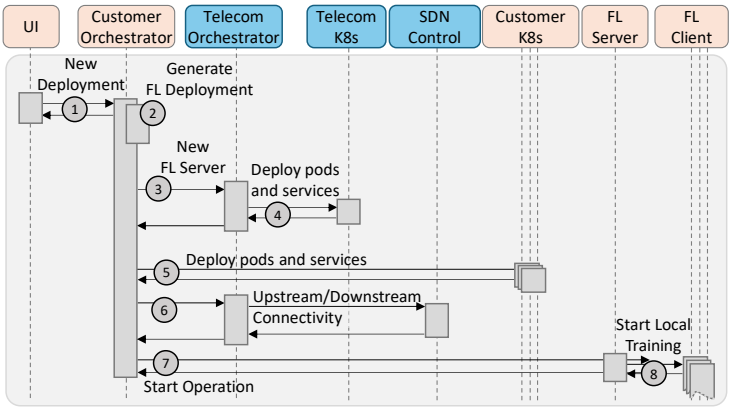

Figure 2 WF1: FL Application Deployment

Let us now define the workflows for the FL application deployment (WF1) and FL application operation (WF2). WF1 (see Figure 2) starts when a customer issues the deployment of a FL application through the user interface (UI) (step 1 in Figure 2). The request is handled by the Customer Orchestrator, which generates a FL deployment descriptor (step 2). The description includes the configuration and resource allocation for each of the components involved in the FL deployment, i.e., FL clients, FL server(s), and upstream and downstream connectivity. Assuming that the FL server needs to be deployed in the telecom cloud, the Customer Orchestrator sends a request to the Telecom Orchestrator for that deployment (step 3). Upon the reception of the request, the Telecom Orchestrator checks it and proceeds with the deployment of the FS server application (step 4). Similarly to the customer cloud, we assume that telecom cloud management is based on Kubernetes. Note that the Telecom Orchestrator makes the decision of the final placement, e.g., based on the computation requirements, available connectivity to the customer locations, etc. Following the deployment of the FL Server, the Customer Orchestrator deploys all the necessary software of the FL clients through each Kubernetes manager in the locations defined on the deployment descriptor (step 5). Next, the Customer Orchestrator requests upstream and downstream connectivity to the Telecom Orchestrator (step 6), which relies on the SDN Controller to configure the actual network nodes. Note that the connectivity between FL nodes needs to support commands and models exchanged between FL clients and the FL server. Therefore, high-capacity is needed when models are exchanged but a minimal capacity needs to be always available. Additionally, FL application upstream's connectivity should be based on unicast individual connections between each FL client and the FL server, whereas downstream connectivity can be based on multicast. Finally, when the deployment is complete, the Customer Orchestrator notifies the FL server to start its operation (7), which in turn notifies the FL clients to start the local training and the deployment phase finishes.

Once the deployment phase has finished, the FL application operation starts and the FL server periodically collects local models from the FL clients, computes a global model, and distributes it back to the FL clients. However, connectivity needs to be managed during these training rounds and the capacity increased and decreased back before and after the models are transferred. Such activity is defined in WF2 (see Figure 3). With a configured periodicity, the FL server starts a training round (step 1 in Figure 3). Then, the FL server sends a request to the Customer Orchestrator to increase the capacity of the upstream connectivity for the FL clients (step 2). The Customer Orchestrator, forwards the request to the Telecom Orchestrator, which in turn relies on the SDN Controller to actually configure the network nodes. Once the FL server receives a confirmation of the capacity allocation, it notifies the selected FL clients to send their local models (step 3). The last version of the local models are sent using the high-capacity upstream connectivity to the FL server (step 4). Once the FL server confirms that all model transfers are completed, it sends a request to the

Customer Orchestrator to decrease back the upstream capacity (step 5) and the FL server computes a new global model (step 6). Before the new global model can be sent to the FL clients, the FL server requests a capacity increase for the downstream connectivity (step 7) and when the capacity update has been confirmed, it notifies all the FL clients (step 8) and distributes the global model (step 9). Once the transfer is completed, the FL server requests to decrease the downstream capacity back to the minimum (step 10) and FL clients use the new global model to continue the local training process (step 11). These steps repeat until the decommissioning of the service or a maximum number of rounds is achieved.

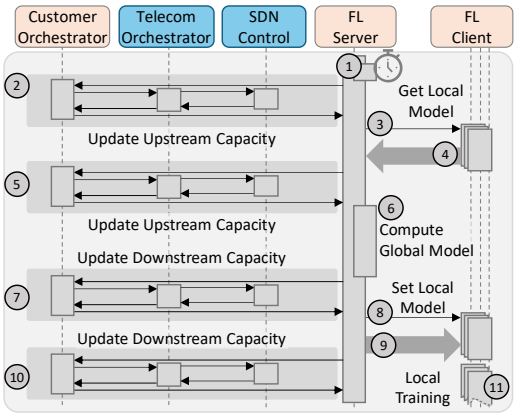

Figure 3 WF2: FL Operation

3. Experimental Results

This section reports the experimental results of the implemented workflows defined in the previous section. The setup used to evaluate the proposed WFs runs in a virtualized environment provided by an OpenStack cluster on-premises. Three different locations have been set up, each one running a Minikube Kubernetes cluster [5]. Minikube offers most of the features found in production-ready Kubernetes clusters and is suited for testing environments due to its simplicity of deployment and maintenance. Moreover, Minikube uses the latest API versions of Kubernetes, making the migration to classic Kubernetes cluster seamless. Each emulated cloud resides in a different VM. The Orchestrators and the SDN Controller have been implemented using Python and are deployed in separate Virtual Machines (VM). The Flower FL framework [6] has been used for evaluation of the FL application. Flower implements both clients and servers, the communication, and the orchestration of the FL training session. Since Flower is agnostic to the ML technology, it allows the user to define their custom client and servers. Besides, Flower offers predefined training strategies defining how the central server should aggregate the data received from the clients. A Web User Interface (UI) has been developed using Flask and eases the interaction of the user with the Customer Orchestrator (see Figure 4). The UI also allows the user to track the status of the deployments and the different steps involved.

Let us start with the assessment of WF1. Figure 5 shows a capture of the exchanged messages of the workflow and follows the same numbering as the one used in Figure 2. The Web UI in Figure 4 shows the configured parameters and the execution log. The deployment phase of the FL application is initiated by the user, who sends a request through the Web UI (message 1 in Figure 5). This request includes the service identifier and a specification of the customer locations where the clients need to be deployed, while leaving the specific server location to be decided by the Telecom Orchestrator. The request is handled by the Customer Orchestrator, which generates the FL deployment descriptor, including the description and configuration of the pods hosting the FL clients. Several services are defined to ensure communication inside and outside the Kubernetes cluster; specifically, for the external communication between the FL components and the Orchestrators, each pod defines a NodePort service (<IP, Port>). The Customer Orchestrator sends a request to the Telecom Orchestrator including the details needed for the deployment of the FL Server (message 3). The Telecom Orchestrator selects the best cluster for the deployment of the FL server, generates a descriptor for the deployment and asks the Telecom Kubernetes manager to allocate the necessary resources on the Telecom Cloud, including the pod and the NodePort service (message 4). Next, the Customer Orchestrator communicates with each Kubernetes manager in the customer locations involved in the deployment of the

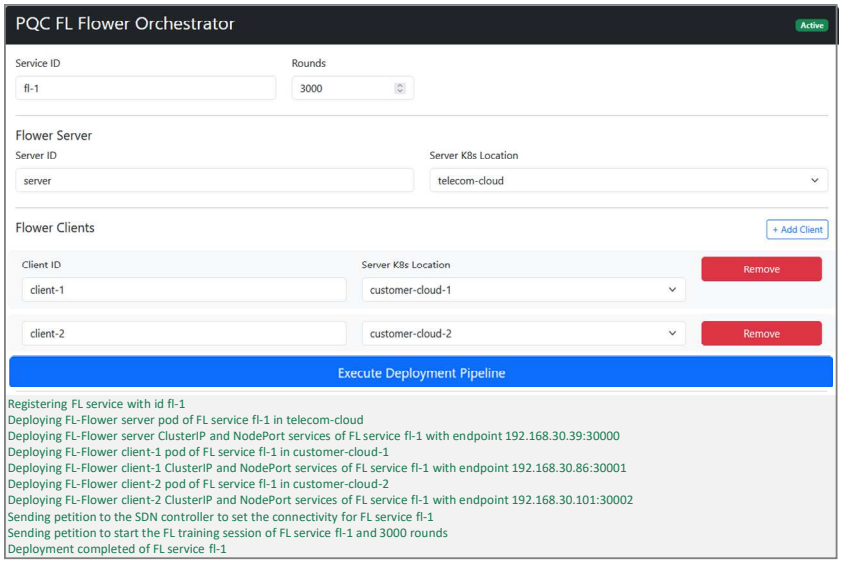

Figure 4 Customer Orchestrator Web User Interface

	Source	Destination	Info
1	UI	Customer Orch.	POST /deploy_fl
	Customer Orch.	UI	HTTP/1.1 202 ACCEPTED
3	Customer Orch.	Telecom Orch.	POST /register_service
	Telecom Orch.	Customer Orch.	HTTP/1.1 201 CREATED
4	Telecom Orch.	Telecom K8s	Application Data
	Telecom K8s	Telecom Orch.	16443 → 54730
5	Customer Orch.	Customer K8s	Application Data
	Customer K8s	Customer Orch.	16443 → 41542
6	Customer Orch.	Telecom Orch.	POST /set_connectivity
	Telecom Orch.	SDN	POST /Setup
	SDN	Telecom Orch.	HTTP/1.1 200 OK
	Telecom Orch.	Customer Orch.	HTTP/1.1 200 OK
7	Customer Orch.	FL Server	POST /Start
	FL Server	Customer Orch.	HTTP/1.1 200 OK
8	FL Server	FL Client	POST /Start
	FL Client	FL Server	HTTP/1.1 200 OK

Figure 5 WF1 Exchanged Messages

FL clients to allocate and deploy the selected services (message 5). The response to the Customer Orchestrator includes the NodePort service for the FL server.

Using the NodePort details of the FL clients and FL server, the Customer Orchestrator sends a request to the Telecom Orchestrator to create the upstream and downstream connectivity of the FL app. (message 6). The specification of the connectivity includes the service id, the NodePort, i.e., the IPs and ports of each of the endpoints exposed by the different FL components, and the specification of the initial capacity for the upstream and downstream connectivity. The Telecom Orchestrator processes the connectivity setup request and forwards it to the SDN Controller which in turn configures the involved network nodes. Finally, the FL training operation can start, so the Customer Orchestrator notifies the FL server to start (message 7), and the FL server notifies the FL clients that the operation can start with the local training (message 8).

Once WF1 finishes, FL operation starts following WF2. Figure 6 shows the exchanged messages. Periodically the FL server runs a FL training round. Before the FL server can retrieve the local models from the FL clients, it must reconfigure the capacity of the uplink connectivity (message 2). The Customer Orchestrator creates a request for the Telecom Orchestrator, which translates the request into a resource reconfiguration that the SDN Controller can handle. Once the updated network capacity is available for the selected links, the FL server sends a notification to each of the FL clients to transfer their local models (message 3). The FL clients transfer the last version of their trained local model to the FL server (message 4). Upon completion, the FL Server has all the required local models from the selected FL clients, so it issues a notification to the Customer Orchestrator to scale down the capacity of the uplinks, which coordinates with the Telecom Orchestrator and the SDN Controller (message 5). Then, the FL server computes a new global model and updates the local models of the selected clients.

	Source	Destination	Info
	FL Server	Customer Orch.	POST /bandwith_reconfig
	Customer Orch.	Telecom Orch.	POST /bandwith_reconfig
	Telecom Orch.	SDN Ctl.	POST /resource_reconfig
2	SDN Ctl.	Telecom Orch.	HTTP/1.1 200 OK
	Telecom Orch.	Customer Orch.	HTTP/1.1 200 OK
	Customer Orch.	FL Server	HTTP/1.1 200 OK
3	FL Server	FL Client	GET /get_local_model
	FL Client	FL Server	HTTP/1.1 200 OK
4	FL Client	FL Server	POST /data
	FL Server	FL Client	HTTP/1.1 200 OK
	FL Server	Customer Orch.	POST /bandwith_reconfig
	Customer Orch.	Telecom Orch.	POST /bandwith_reconfig
	Telecom Orch.	SDN Ctl.	POST /resource_reconfig
5	SDN Ctl.	Telecom Orch.	HTTP/1.1 200 OK
	Telecom Orch.	Customer Orch.	HTTP/1.1 200 OK
	Customer Orch.	FL Server	HTTP/1.1 200 OK
	FL Server	Customer Orch.	POST /bandwith_reconfig
	Customer Orch.	Telecom Orch.	POST /bandwith_reconfig
	Telecom Orch.	SDN Ctl.	POST /resource_reconfig
7	SDN Ctl.	Telecom Orch.	HTTP/1.1 200 OK
	Telecom Orch.	Customer Orch.	HTTP/1.1 200 OK
	Customer Orch.	FL Server	HTTP/1.1 200
8	FL Server	FL Client	GET /set_local_model
	FL Client	FL Server	HTTP/1.1 200 OK
9	FL Server	FL Client	POST /data
	FL Client	FL Server	HTTP/1.1 200 OK
	FL Server	Customer Orch.	POST /bandwith_reconfig
	Customer Orch.	Telecom Orch.	POST /bandwith_reconfig
	Telecom Orch.	SDN Ctl.	POST /resource_reconfig
10	SDN Ctl.	Telecom Orch.	HTTP/1.1 200 OK
	Telecom Orch.	Customer Orch.	HTTP/1.1 200 OK
	Customer Orch.	FL Server	HTTP/1.1 200 OK

Figure 6 WF2 Exchanged Messages

4. Concluding Remarks

Coordinated orchestration and operation of FL applications deployed over hybrid customer–telecom cloud infrastructures have been proposed and experimentally assessed. The main objective was to reduce the connectivity utilization required to exchange models during FL training. The solution enables capacity-on-demand operation at each training round. Experimental validation demonstrated the feasibility of deploying FL workloads and reconfiguring network connectivity using cloud-native technologies for workload management and SDN for transport network control.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Union's Horizon Europe research and innovation programme under G.A. No. 101092766 (ALLEGRO), the project PID2024-157824OB-I00 funded by MICIU/ AEI/10.13039/501100011033 /FEDER, UE and from ICREA.

REFERENCES

[1] B. McMahan *et al*., "Comm.-Efficient Learning of Deep Networks from Decentralized Data," in Proc. AISTATS, 2017.
[2] S. Matta *et al*., "FL for Privacy-Preserving Healthcare Data Sharing," Am. J. of Scholarly Research and Innovation, 2025.
[3] M. Isaksson and K. Norrman, "Secure Federated Learning in 5G Mobile Networks," in Proc. GLOBECOM 2020.
[4] C. Wu *et al*., "Communication-efficient federated learning via knowledge distillation," Nature Communications, vol. 13, 2022.
[5] Minikube [On-line] https://minikube.sigs.k8s.io/docs/
[6] Flower FL Framework [On-line] https://flower.ai/